- ARRAY–

Array: A set of the same data type where multiple data can be used with one name. A collection of several variables.

- The size of arrangement is set in the beginning and **cannot be changed**
- ORDER: Declare array → allocate memory for an array → use an array

$$\underbrace{\texttt{int []varName}}_{\text{Arrangement declaration (배열 선언)}}\ \texttt{= new int}\ \underset{\substack{\text{Length}\\ \text{Index order = length -1}}}{\texttt{[]}}$$

1. **OTHER FORMS:**

1.1 Array type string:

```
String []varName = new String []
```

1.2 Array type int with values prescribed:

```
int [] varName = {num1, num 2, num 3…}
```

1.3 Array type objName with multiple values prescribed:

```
objName[] varName = {new objName (value1),
                     new objName (value2),
                        new objName2 (value3, value4),
                           new objName2 (value5, value6)};
```

2. **INSERTING VALUES IN THE ARRAY:**

2.1 Integer

```
    int [] Arr = new int[5];
for (int i = 0 ; i < Arr.length ; i++){
           Arr[i] = i+1;
    System.out.println(Arr[i]); }
```

```
output:
1
2
3
4
5
```

2.2 String

```
  String [] Arr2 = {"A", "B", "C"};
for (int i = 0 ; i < Arr2.length ; i++){
    System.out.println(Arr2[i]);}
```

```
output:
A
B
C
```

3. MULTIDIMENTIONAL ARRAY

열 (Row)

행 (Column)

(0,0)	*(0,1)*	*(0,2)*
(1,0)	*(1,1)*	*(1,2)*
(2,0)	*(2,1)*	*(2,2)*

3 x 3 의 행렬

```
int [][] matrix = new int [3][3];

matrix [0][0] = 1; matrix [0][1] = 2;
matrix [0][2] = 3; matrix [1][0] = 4;
matrix [1][1] = 5; matrix [1][2] = 6;
matrix [2][0] = 7; matrix [2][1] = 8;
matrix [2][2] = 9;

                OR

int [][] matrix = {{1,2,3},
                   {4,5,6},
                   {7,8,9}};
```

열 (Row)

행 (Column)

1	2	3
4	5	6
7	8	9

3 x 3 의 행렬

1. **COPYING ARRAY**

Method 1:	**Method 2:**	**Method 3:**
Copying array 'score' to a new variable named 's' using the ***<u>for command</u>***	Copying array 'score' to a new variable named 's' using *the* ***<u>array copy</u>* <u>class</u>.**	Copying array 'score' to a new variable named 's' using *the*

	In System.arraycopy: - score: original variable - 0: the initializing index value - s: copied variable - 0: from which index (in s) will it insert the copied values - score.length: the length of the array that is to be copied.	***Copyof() method*** *from the class Arrays.*

`int[] score = {100,10,20,30,40};`
`int[] s = new int[score.length];`
 	
`for(int idx=0 ; idx<score.length ;`
`idx++) {`
`s[idx] = score[idx];`
`}`
`System.out.printf("score[%d]=%d \t`
`s[%d]=%d\n", idx, score[idx], idx,`
`s[idx]);`
`         }`
	`int[] score = {100,10,20,30,40};`
	`int[] s = new      int[score.length];`
	
	`System.arraycopy(score, 0, s, 0,`
	`score.length);`
	
	`for(int idx=0 ; idx<score.length ;`
	`idx++) {`
	`System.out.printf("score[%d]=%d \t`
	`s[%d]=%d\n", idx, score[idx], idx,`
	`s[idx]);`
	`         }`
		`int[] score = {100,10,20,30,40};`
		`int[] s = null;`
		
		`s = Arrays.copyOf(score, score.length);`
		`for(int idx=0 ; idx<s.length ; idx++) {`
		`System.out.printf("score[%d]=%d \t`
		`s[%d]=%d\n",`
		
		`idx, score[idx],idx,s[idx]);`
		`         }`

Output:
`score [0] = 100    s[0] = 100`
`score [1] = 10      s[1] = 10`
`score [2] = 20      s[2] = 20`
`score [3] = 30      s[3] = 30`
`score [4] = 40      s[4] = 40`		

2. IMPROVED FOR COMMAND

- Allows the program to read the command but not change in (increment or decrement it). Starting from the value in the first array, each array value gets printed in a variable defined by the programmer.
- Example 1: 65 is inputted in the variable "number" and this is printed, then 235 is inputted in the variable "number" and this is printed…etc…)
- For the improved for command, you cannot change a value in the array, whereas with the normal for command, you can. (See example 2)

Example 1:	Example 2:

`int[] array = {65,235,32,58,92};`
`for (int number : array) {`
`System.out.print(number + "\t" );`
	`int[] arr = new int[3];`
	`        for(int i=0 ; i<arr.length ; i++) {`
	`        arr[i] = 5;}`
	
	`        for(int temp : arr) {`
	`        temp = 9;}`
	
	`        for(int temp : arr) {`
	`        System.out.println(temp);`

Output: 65 235 32 58 92	*Output:* 5 5 5

3. REFERENCE TYPE

Primitive Type: Store literal value of an integer, a real number, or a literal data type

```
String name1 = "Amy";
```

Reference Type: Does not store literal value but stores the address of where the literal value is stored.

```
String name1 = new String("Amy");
```

- When a reference type variable is created, the variable will have a null value
- When you make a variable with the same address **`(int[] variable1 = variable2;)`** and change something in either variable, the other variable also changes value in the same way, as they share the same address. (if an array is copied, they <u>do not share the same address</u>).

```
int [] arr1 = {1,2,3}
int [] arr3 = arr1;
arr3[0] = 4;
System.out.print(arr3[0] + “\t”);
System.out.print(arr1[0]);
```

```
Output: 4   4
```

Comparing memory address:	Comparing the actual String value:
`String name1 = new String(“Amy”);` `String name2 = new String(“Amy”);` `System.out.println(name1 == name2);`	`String name1 = new String(“Amy”);` `String name2 = new String(“Amy”);` `System.out.println(name1.equals(name2));`
false	*true*